Prévisions de travail sur la proposition de stage de publications R3MOB

Cette mission vise à automatiser l'actuelle gestion des publications scientifiques sur le site *R3MOB*. En effet, ces dernières sont actuellement rentrées manuellement.

L'utilisateur est confronté principalement à deux problèmes lors de sa saisie :

- l'importation de la publication.
- la classification de cette dernière en *thématiques* et en *sous thématiques scientifiques*.

La section 1 vise à décrire comment automatiser ces problèmes, la seconde vise à intégrer la solution proposée en section 1 dans le temps imparti.

Moyens techniques

Comment récupérer les métadonnées de la publication en restant user friendly?

Comment favoriser une exportation de la base de données?

Comment catégoriser les publications?

Récupération des publications scientifiques

En pratique, je suppose que l'utilisateur n'aura pas besoin d'importer réellement la publication. L'intérêt n'est pas de stocker cette dernière mais bien de la retrouver sur internet, car je suppose qu'elle a bien été publiée quelque part. Je pense que les *métadonnées* récupérées sur les sites devraient être toutes converties en *json* pour normaliser, et ainsi faciliter l'insertion dans la base de données.

Voici les différents moyens possibles pour l'utilisateur d'importer sa publication :

1. (**** important) l'utilisateur fournit le *DOI*, ce qui serait idéal pour le développeur! En effet, il semble qu'une publication scientifique possède un identifiant unique.

Voici les solutions possibles :

- Api *Crossref* : publique, gratuite et open source!

Néanmoins, la documentation de cette api semble être un peu *deprecated* à en voir leur github.

Je pense qu'un simple script *Python* ou *Javascript* pourrait suffire pour interagir avec cet api. Elle serait même utilisable seulement avec des requêtes *Curl*.

- Api *Semantic Scholar* : moins complète que *Crossref* mais elle inclut des statistiques sur les citations!

Je choisirais quand même *Crossref*, mais *Semantic Scholar* pourrait fournir des statistiques intéressantes... L'idée est de récupérer un *json* comportant les métadonnées de la publication, *URL* compris.

INCONVENIENTS SOLUTION: documentation un peu *deprecated*, est ce que l'utilisateur aura le *DOI* de la publication? **AVANTAGES SOLUTION**: très facile d'utilisation, gratuit, open source.

2. (* pas important) L'utilisateur copie-colle l'URL de la publication, ce serait idéal pour l'utilisateur! Ce n'est pas le *DOI* de la publication, mais on s'en rapproche.

L'idée est d'analyser l'URL pour identifier le site de provenance parmis *Google Scholar*, *HAL* etc..

Ensuite on pourrait utiliser l'API correspondante au site de provenance, pour retrouver ensuite le *DOI* de la publication et faire écho à la solution 1:

- Pour *HAL*, c'est facile! L'utilisateur donne `https://arxiv.org/abs/2203.12345` et le développeur devine `https://export.arxiv.org/api/query?id_list=2203.12345` .

Un simple script *Python* pourrait être utilisé ici.

INCONVÉNIENTS: Je me suis renseigné sur *Google Scholar*, il semble qu'il ne possède pas d'api. De plus cette approche dépend du site source. Possibilité d'attaque **CSRF** ou autre injection je suppose.

AVANTAGES: user-friendly

3. (** moyennement important) L'utilisateur fournit un fichier *BibTex*.

Cette technique pourrait être utilisée par un utilisateur car les sites comme *Google Scholar* le proposent.

Je pense que la meilleure idée est de tout convertir en *json*, *Bibtex* compris.

Exemple:

```
@INPROCEEDINGS{10770481,
  author={Yacheur, Badreddine Yacine and Anamuro, Cesar Vargas and Dehbi, Moad and Bouzaidi Tiali, Mohamed Amine and Mosbah, Mohamed
  booktitle={2024 20th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)},
  title={Enhancing Vehicle Orientation in Toll Stations Using vMEC and Hybrid Vehicular Communications},
  year={2024},
  volume={},
  number={},
  pages={1-7},
  keywords={Cellular networks;Wireless communication;Unicast;Scalability;Europe;Computer architecture;Reliability;Servers;Security;E
  doi={10.1109/WiMob61911.2024.10770481}}
```

On parse en *json* pour normaliser le format qui sera ensuite donné à la base de données:

```
{
  "type": "inproceedings",
  "id": "10770481",
  "author": [
    "Yacheur, Badreddine Yacine",
    "Anamuro, Cesar Vargas",
    "Dehbi, Moad",
    "Bouzaidi Tiali, Mohamed Amine",
    "Mosbah, Mohamed",
    "Bonnin, Jean-Marie",
    "Ayaida, Marwane",
    "Ahmed, Toufik"
  ],
  "booktitle": "2024 20th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)",
  "title": "Enhancing Vehicle Orientation in Toll Stations Using vMEC and Hybrid Vehicular Communications",
  "year": 2024,
  "volume": null,
  "number": null,
  "pages": "1-7",
  "keywords": [
    "Cellular networks",
    "Wireless communication",
    "Unicast",
    "Scalability",
    "Europe",
    "Computer architecture",
    "Reliability",
    "Servers",
    "Security",
    "Edge computing",
    "Hybrid vehicular network",
    "vMEC",
    "ITS-G5",
    "C-V2X",
    "Toll management",
    "C-ITS testbed"
  ],
  "doi": "10.1109/WiMob61911.2024.10770481"
}
```

Cette conversion peut se faire en un simple script *Python* encore une fois.

INCONVÉNIENTS: le format, je veux du *json*.

AVANTAGES: Toutes les informations sont présentes, c'est toujours mieux que de devoir faire des requêtes à une api.

2. (**) Importation par fichier *XML*. On fait pareil que pour *BibTex*.
3. (**** Important++) L'utilisateur donne déjà un fichier *json* contenant toutes les informations de la publication.

Ce fichier *json* doit suivre une règle de formatage précise, car il doit contenir les bons mots clés. Le mot clé *"DOI"* ne doit pas être écrit *"DOY"*. Je pense que je peux abstraire l'effet *case sensitive*, les majuscules ou minuscules dans les mots clés importent peu.

INCONVÉNIENTS: suit une règle de formatage stricte.

AVANTAGES: application friendly, le *json* serait la norme. Insertion facile dans la *bdd*.

4. (* pas important): l'utilisateur effectue une recherche manuelle. Il entre un *titre* et un *auteur*.

Il suffit ensuite d'utiliser la méthode 1 avec la *Crossref* Api pour récupérer les métadonnées de la publication, si elle existe, et ensuite de formater en *json* le résultat avec la méthode 3.

==> Je pense qu'il faut pouvoir transformer les métadonnées trouvées par les différentes méthodes proposées en *json*, pour faciliter ensuite l'insertion dans la *bdd*. Cela nécessitera des scripts de *parsing* en *Python*.

Il y aurait donc au moins 3 sous-modules :

1. Module qui s'occupe d'interagir avec *Crossref* api, ou les autres *api* si la méthode 2 est implémentée.
2. Module qui s'occupe de convertir tout type de fichier en *json*.
3. Module qui s'occupe d'insérer le *json* associé à la publication dans la *bdd*.

Insertion dans la base de données

Si j'étais stagiaire, j'utiliserais *PostgreSQL* car j'ai déjà effectué par le passé un projet avec. Ce projet portait sur l'automatisation de la vente et achat de pièces et boîtes légos.

1. Je pense qu'il serait intéressant d'utiliser plusieurs tables comme :

- *publications* (***)
- *authors* (***)
- *keywords* (**)
- *themes* (*)
- *subthemes* (*)

En effet, plusieurs auteurs pourraient être à l'origine d'une unique publication et inversement.

2. Ensuite, il faudrait pouvoir trouver un moyen de *sécuriser* cette base de données pour éviter tout problème de publication sans nom ou sans auteur. Il faut respecter des contraintes d'intégrité qu'il faudra définir, sous la forme de *triggers*, et il en faut le plus possible. Cette étape pourra prendre du temps et devra aussi nécessiter une certaine phase de tests.

3. Pouvoir exporter la base de données pour le confort de l'utilisateur, et des potentiels tests de bout en bout. Je suppose que vous voudriez pouvoir transporter votre base de données avec vous. Je crois d'ailleurs que vous donnez des fichiers *json* à *TheBrain*, c'est pourquoi cette fonctionnalité de convertir la *bdd* en un ensemble de *json* doit être implémentée. Je suppose qu'il est possible d'utiliser les fonctions inhérentes de *PostgreSQL* pour réaliser cette conversion, à voir.

En terme de sécurité, je ne sais pas encore comment les insertions seront faites dans la *bdd*. Il est possible que des ports doivent être ouverts. Néanmoins, je pense qu'il serait important de conteneuriser cette *bdd* avec *docker* en *rootless*. Ceci et les *triggers* de la *bdd* devraient suffire à empêcher toute injection non désirée, ou en tout cas à les ralentir.

Je me demande aussi s'il serait intéressant de pouvoir chiffrer certaines parties de la *bdd*, la *table* des auteurs par exemple. A voir.

Catégorisation des publications scientifiques

Les publications scientifiques sont maintenant stockées dans la base de données, sous formes de métadonnées et de liens vers de potentielles *api*s qui pourraient être utilisées pour récupérer encore plus d'informations sur ces dernières.

Comment éviter la catégorisation manuelle?

Comment catégoriser ces publications selon cette architecture de stockage prédéfinie?

Comment allier analyse sémantique et analyse par mots clés?

Serait-il possible d'ajouter automatiquement des catégories?

Nous supposons dorénavant que les mots clés, le titre et le résumé de la publication sont présents dans la *bdd*, ou accessibles par api. Voici les différentes méthodes utilisables pour classifier cette base de données.

1. (* pas important mais pourrait être utilisé pour la première release): filtrer les publications selon les mots-clés, dans le titre, l'abstract ou ceux trouvés par l'api.

L'idée ici est de construire une base de mots clés pour chaque sous catégorie. Par exemple, pour le thème "énergie renouvelable", on peut penser à "transition écologique" comme sous thématique.

- Il faut d'abord définir un dictionnaire de règles prédéfini. Par exemple pour les thèmes : `"logistique" donne "supply chain" ou "mobilité"`. Par exemple pour les sous thématiques : `["marketing", "supply chain", "logistique"] donne "marketing", "supply chain" et "logistique"`.

- on effectue une boucle *for* sur tous les mots de l'*abstract*, des mots clés, et des mots du titre de la publication, puis on applique les règles associées.

Si le titre contient *logistique* alors on insère dans la *bdd* le thème *logistique* pour cette publication.

Je suppose qu'il est possible d'encore utiliser un script *Python* pour implémenter cette idée, avec le module *psycopg2* pour interagir avec la *bdd PostgreSQL*.

INCONVÉNIENTS: les thématiques ne sont pas forcément toutes détectées, il faut qu'elles soient aussi prédéfinies. Elle ne capture pas le contexte des phrases écrites dans l'abstract. La complexité temporelle est polynomiale selon le nombre de règles.

AVANTAGES: simple et rapide à faire, parfait pour la première release. Les règles sont ajustables facilement. Ne nécessite pas forcément d'une base de connaissances.

2. (*** important): amélioration de la flexibilité de la méthode 1 avec un classificateur automatique.

Après réflexion, je pense que l'intérêt de cette mission n'est pas d'innover sur une façon de classifier des données. Dans mes recherches, je suis tombé sur le site internet `elastic.io`, nommé *elasticsearch*. Ce site semble implémenter une méthode à base de, je cite “TF-IDF similarité Cosine" pour automatiser les recherches. Cette méthode permet de mesurer à quel point un mot est commun ou rare pour déterminer son importance.

Il semble qu'il suffise d'utiliser des modules de la librairie *Panda* de *Python* qui implémente cette méthode. Néanmoins je ne sais pas encore comment il serait possible de le tester. Il serait possible de faire des comparaisons entre l'actuelle version sur le site *R3MOB*, rentrée manuellement, et la version implémentée de *TD-IDF*.

Il semble aussi que cette méthode ne fonctionne que si l'on fournit au préalable des textes descriptifs pour chaque thème et sous thématique scientifique.

INCONVÉNIENTS: nécessite une base de connaissance (texte descriptif par thème), nécessite la familiarisation avec le module *Panda* de *Python*implémentable que comme ça. Le coût en calcul sera plus grand, il faut que je me renseigne. **AVANTAGES**: précis(à tester), modulable car des algorithmes de *clustering* peuvent être utilisés par dessus grâce à *Panda*, assez facilement.

3. (* pas important) Utilisation d'un modèle *NLP* : *Natural Language Processing*.

Cela utilise un modèle de langage statistique pour comprendre la signification d'une phrase. Il gère aussi les synonymes et variations lexicales. L'apprentissage se fait en continu.

C'est très difficile à implémenter, je n'ai pas trouver de réelle documentation sur ce sujet mis à part le modèle de langage *BERT*. Cette solution serait la meilleure de loin, mais nécessiterait de s'y pencher dessus beaucoup de temps.

INCONVÉNIENTS: difficile à implémenter, chronophage, difficile à comprendre (boîte noire). Nécessite une quantité énorme de calculs je suppose.

AVANTAGES: c'est la mode en ce moment donc il est possible de trouver des scripts déjà faits, très puissant.

==> Le meilleur algorithme serait *TF-iDF* avec similarité *cosine*. L'implémentation pourrait prendre du temps, c'est pourquoi la première release pourrait d'abord être une sorte de correspondance de règles simples (méthode 1).

De plus, tout se ferait avec *Python*, encore. Je n'aime personnellement pas le *java* en général parce que cela nécessite tellement de ressources pour faire tourner un service sur un serveur. Cela se compterait en plusieurs Go de *Ram* pour ce projet, s'il était fait en *java*.

Cela nécessiterait au moins 2 modules :

1. Gestion des catégories, par correspondance simple d'abord puis par modèle d'IA à la fin.
2. Insertion dans la base de données avec *Python* et *psycopg2*.

De plus, il serait possible d'allier les deux méthodes de recherche par mot clé et par *TF-IDF* en enrichissant les textes donnés à l'algorithme *TD-IDF* avec les mots-clés récupérés depuis une api, ou depuis la *bdd*.

Moyens organisationnels

Ainsi, cette mission se découpe en trois grandes parties. Chaque partie peut être réalisée assez facilement. Néanmoins beaucoup d'améliorations sont possibles, et même peut être exigées.

Les différentes releases possibles

0. Petite Release : Prend au maximum une semaine --> faire un état de l'art sur les outils de chargement de publications scientifiques.
1. Première release : (début stage + petite release) à (1 mois après début stage) --> fournir une première version utilisable.
2. Deuxième release : (1ere release à (1 mois après 1ère release) --> implémenter toutes les façons d'importation, sécuriser la *bdd* au maximum, ajout de règles de catégorisation. Effectuer les *fix* proposés par celui qui fera le *frontend*.
3. Troisième release : (2ème release à (1 mois après 2ème release) --> implémenter *TF-IDF* + similarité *Cosine* pour la catégorisation. Effectuer des tests d'envergure, en considérant une *bdd* que l'on sait bien triée. Effectuer les *fix* proposés par celui qui fera le *frontend*.
4. Dernière release : (3ème release) à fin stage --> implémenter un *NLP*? Utiliser *BERT*? Affinage de *TF-IDF* avec de l'enrichissement par mots clés. Insertion interactive avec recherche par titre et auteur, appel *Crossref* api intelligent. Effectuer les *fix* proposés par celui qui fera le *frontend*.

Cette organisation me paraît un peu déséquilibrée, mais elle donne un vue d'ensemble plutôt correcte de la mission.

Ébauche de diagramme de gantt

1. Première release

liste des tâches:

Récupération.

- définir une règle de formatage pour les fichiers *json*.
- abstraire l'effet *case-sensitive* dans les fichiers *json*.
- gérer les erreurs si *json* pas complet. La récupération des éléments manquants se fera plus tard.
- implémenter méthode pour parser en *json* des fichiers *Bibtex*.
- implémenter méthode pour parser en *json* des fichiers *XML*.
- implémenter un convertisseur *json* to *sql*.

Base de données.

- définir le schéma conceptuel.
- définir le schéma relationnel.
- définir les contraintes d'intégrité.
- implémenter le schéma relationnel.
- implémenter les contraintes d'intégrité. Pas d'écriture de triggers de sécurisation pour l'instant.
- exportation de la *bdd* en *json* pour l'utiliser dans *TheBrain*.

Catégorisation.

- définir les thèmes nécessaires.
- définir les sous thèmes scientifiques nécessaires.
- écriture de règles.
- récupérer l'abstract avec *api*. Code temporaire car module *api* pas implémenté pour l'instant.
- implémentation de la méthode 1, filtrer les publications par mots-clés.
- remplir la *bdd* avec les thèmes et sous thèmes scientifiques trouvés.

2. Deuxième release

Liste des tâches:

Récupération.

- implémenter le module *api*, pour *google scholar*, *Crossref*, etc.. Toutes les *api*.
- implémenter la version user friendly qui met juste un *URL*.

Base de données.

- écriture de *triggers* le plus possible.
- conteneurisation de la *bdd*.
- je suis sur que celui qui fera le front-end aura des problèmes avec l'interaction de la *bdd*, l'aider.

Catégorisation.

- ajout de règles de catégorisation intéressantes.
- amélioration de la méthode 1, elle est très modulable, je sais pas encore comment.

3. Troisième release

Liste des tâches:

Récupération.

- rien je pense.

Base de données.

- effectuer les *fix* nécessaires par celui qui fera le frontend.

Catégorisation.

- convertir l'ancien algorithme en quelque chose d'utilisable par *TF-IDF*.
- implémenter *TF-IDF* + similarité *Cosine*
- trouver une *bdd* de publications pour tester le modèle implémenté.
- tester le modèle.
- proposer des méthodes d'amélioration du modèle par *clustering*.

4. Dernière release

Liste des tâches:

Récupération.

- insertion intéractive par recherche de titre ou auteur, appel aux *bdd* des sites comme *Google Scholar* en temps réel avec l' *api*.

Base de données.

- effectuer les *fix* nécessaires par celui qui fera le frontend.

Catégorisation.

- Affinage de *TF-IDF* avec de l'enrichissement par mots clés.
- implémenter un *NLP*?

Je me dis aussi qu'il va falloir écrire des documents de spécifications techniques et spécifications fonctionnelles. Il faudrait l'intégrer quelque part entre la release 1 et la release 2.

Je ne sais pas non plus comment fournir les éléments de la bdd au frontend. Je pense qu'il va aussi falloir utiliser un framework comme *Nodejs* à un moment pour coder des fonctions d'intéraction entre la *bdd* et le frontend.

EOF